Data Science and AI for Business

Lecture 0: Math and Python Foundations

Dr. (James) Yunying Huang

Fordham Gabelli School of Business

February 13, 2026

Lecture Outline

Why Math and Python Foundations?

Every method in this course rests on four pillars:

Linear Algebra

- Vectors in $\mathbb{R}^d$
- Similarity = dot product
- Embeddings = matrices
- SVD = dim. reduction

Calculus & Optim.

- Training = optimization
- Gradients $\rightarrow$ direction
- Chain rule $\rightarrow$ backprop
- Loss functions

ML Foundations

- Supervised learning
- Train / val / test
- Evaluation metrics
- Representation learning

Python / NumPy

- Implementation tool
- Vectorized compute
- gensim, PyTorch
- Reproducible workflows

Goal

Build fluency so that next lecture (Word2Vec) focuses on **concepts**, not notation.

Scalars, Vectors, and Matrices

Scalar: a single number

$$x = 3.14$$

Example: a stock return $r = 0.05$

Vector: an ordered list

$$\mathbf{v} = \begin{pmatrix} 0.2 \\ -0.4 \\ 0.7 \end{pmatrix} \in \mathbb{R}^3$$

Example: a word embedding

Matrix: a 2-D array

$$W = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \in \mathbb{R}^{3\times 2}$$

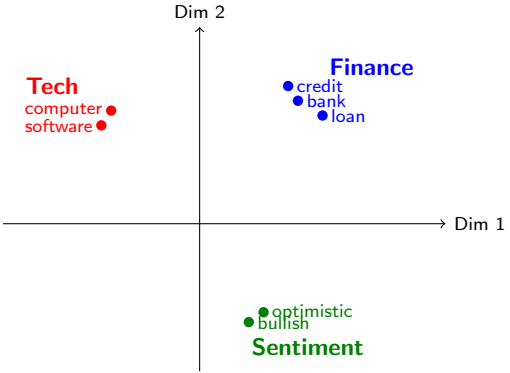
Example: an embedding matrix

Notation conventions:

- Scalars: lowercase italic (x, y, α) Vectors: bold lowercase $(\mathbf{v}, \mathbf{u}, \mathbf{w})$
- Matrices: uppercase $(\mathbf{W}, \mathbf{X}, \mathbf{U})$ $W \in \mathbb{R}^{m\times n}$: m rows, n columns

Why Vectors? The Embedding Intuition

A word embedding maps each word to a point in $\mathbb{R}^d$



Key idea: Similar words → nearby vectors. "Meaning" is encoded in **direction** and **position**.
In real models: $d = 100$–300 dimensions (not just 2).

Practice: Vectors and Matrices

Try these on your own (2 minutes):

1. Given $\mathbf{v} = (0.3,\ -0.1,\ 0.8,\ 0.5)$, what is the dimensionality d of $\mathbf{v}$? What space does it live in?
2. An embedding matrix $W \in \mathbb{R}^{300 \times 50{,}000}$ stores embeddings for a 50,000-word vocabulary. How many total numbers are stored in W?
3. Write down the 2×3 matrix A whose entry $A_{ij} = i + j$.

Solution: Vectors and Matrices

1. $\mathbf{v} \in \mathbb{R}^4$, so $d = 4$. Each component could represent one dimension of a word embedding.
2. $300 \times 50{,}000 = 15{,}000{,}000$ parameters (15 million numbers). This is the size of a typical Word2Vec model.
3. Entry $A_{ij} = i + j$:

$$A = \begin{pmatrix} 1+1 & 1+2 & 1+3 \\ 2+1 & 2+2 & 2+3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix} \in \mathbb{R}^{2\times 3}$$

Matrix Multiplication

Rule: $C = AB$ where $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p} \Rightarrow C \in \mathbb{R}^{m \times p}$
Each entry: $C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$ (row of A · column of B)

Example:

$$\underbrace{\begin{pmatrix} 1 & 0 & 2 \\ 3 & 1 & -1 \end{pmatrix}}_{2\times 3} \underbrace{\begin{pmatrix} 2 & 1 \\ 0 & -1 \\ 4 & 3 \end{pmatrix}}_{3\times 2} = \underbrace{\begin{pmatrix} 1\cdot 2 + 0\cdot 0 + 2\cdot 4 & 1\cdot 1 + 0\cdot(-1) + 2\cdot 3 \\ 3\cdot 2 + 1\cdot 0 + (-1)\cdot 4 & 3\cdot 1 + 1\cdot(-1) + (-1)\cdot 3 \end{pmatrix}}_{2\times 2} = \begin{pmatrix} 10 & 7 \\ 2 & -1 \end{pmatrix}$$

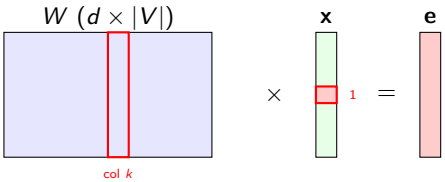
Dimension Compatibility

Inner dimensions must match: $(m \times \mathbf{n}) \times (\mathbf{n} \times p)$. If they don't match, the multiplication is undefined.

Matrix Multiplication: Why It Matters for NLP

Embedding lookup is matrix-vector multiplication:
Let $W \in \mathbb{R}^{d \times |V|}$ be the embedding matrix (d = embedding dim, $|V|$ = vocab size). Let $\mathbf{x}$ be a one-hot vector for the word "bank" (all zeros except position k).

$$\mathbf{e} = W\mathbf{x} = \text{column } k \text{ of } W = \text{embedding of "bank"}$$



All of deep learning = sequences of matrix multiplications + nonlinear functions.

Practice: Matrix Multiplication

Try these on your own (3 minutes):

1. $A \in \mathbb{R}^{2\times 3}$, $B \in \mathbb{R}^{3\times 4}$. What is the shape of AB? Can you compute BA?
2. Compute: $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \end{pmatrix}$
3. If $W \in \mathbb{R}^{d\times|V|}$ and $\mathbf{x}$ is a one-hot vector with $x_3 = 1$ (all others zero), what is $W\mathbf{x}$?

Solution: Matrix Multiplication

1. $AB \in \mathbb{R}^{2\times 4}$ (inner dims match: $\mathbf{3} = \mathbf{3}$). BA is **undefined**: $B \in \mathbb{R}^{3\times 4}$, $A \in \mathbb{R}^{2\times 3} \Rightarrow$ inner dims $4 \neq 2$.
2. Row-by-column:
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 1 \times 5 + 2 \times 6 \\ 3 \times 5 + 4 \times 6 \end{pmatrix} = \begin{pmatrix} 17 \\ 39 \end{pmatrix}$$
3. $W\mathbf{x}$ = column 3 of W = the embedding vector for the 3rd word in the vocabulary. One-hot multiplication selects a single column.

The Dot Product: Core Operation of NLP

Dot product of $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$: $\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v} = \sum_{i=1}^{d} u_i v_i$

Numerical example:

$$\mathbf{u} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}, \quad \mathbf{u} \cdot \mathbf{v} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

Geometric interpretation: $\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos\theta$

- θ small (similar direction): dot product is **large and positive**
- $\theta = 90^\circ$ (orthogonal): dot product is **zero**
- θ large (opposite direction): dot product is **negative**

In Word2Vec

The dot product $\mathbf{u}_o^T \mathbf{v}_c$ measures how likely context word o is given center word c.

Practice: Dot Product

Try these on your own (2 minutes):

1. Compute $\mathbf{u} \cdot \mathbf{v}$ for $\mathbf{u} = (2, -1, 3)$, $\mathbf{v} = (1, 4, -2)$.
2. If $\mathbf{u} \cdot \mathbf{v} = 0$, what does this tell us about the angle between $\mathbf{u}$ and $\mathbf{v}$?
3. In Word2Vec, if $\mathbf{u}_o^T \mathbf{v}_c = 5.2$, does this suggest word o is a likely or unlikely context word for center word c?

Solution: Dot Product

1. $\mathbf{u} \cdot \mathbf{v} = 2 \times 1 + (-1) \times 4 + 3 \times (-2) = 2 - 4 - 6 = -8$.
   Negative value $\Rightarrow$ vectors point in roughly opposite directions ($\theta > 90°$).
2. $\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta = 0 \;\Rightarrow\; \cos\theta = 0 \;\Rightarrow\; \theta = 90°$.
   The vectors are **orthogonal** (perpendicular). In embedding space, this means the two words have no linear relationship along these dimensions.
3. **Likely.** A high dot product $\mathbf{u}_o^T \mathbf{v}_c = 5.2$ means these vectors are well-aligned. After applying softmax, this becomes a high probability $P(o \mid c)$.

Vector Norms: Measuring Magnitude

The norm $\|\mathbf{v}\|$ measures the "length" of a vector.

L_2 Norm (Euclidean):

$$\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{d} v_i^2} = \sqrt{\mathbf{v}^T\mathbf{v}}$$

Most common in NLP.
Used for cosine similarity denominator.

L_1 Norm (Manhattan):

$$\|\mathbf{v}\|_1 = \sum_{i=1}^{d} |v_i|$$

Used in L_1 regularization (Lasso), which encourages sparsity.

Example: $\mathbf{v} = (3, 4)$

- $\|\mathbf{v}\|_2 = \sqrt{9+16} = 5$ $\quad\quad$ $\|\mathbf{v}\|_1 = 3 + 4 = 7$

Unit vector: $\hat{\mathbf{v}} = \mathbf{v}/\|\mathbf{v}\|_2$ has length 1. Normalizing strips magnitude, preserving only **direction**.

Practice: Vector Norms

Try these on your own (2 minutes):

1. Compute $\|\mathbf{v}\|_2$ and $\|\mathbf{v}\|_1$ for $\mathbf{v} = (1, -2, 2)$.
2. Normalize $\mathbf{v} = (1, -2, 2)$ to a unit vector $\hat{\mathbf{v}}$.
3. If $\|\mathbf{v}\|_2 = 0$, what can you conclude about $\mathbf{v}$?

Solution: Vector Norms

1. $\|\mathbf{v}\|_2 = \sqrt{1^2 + (-2)^2 + 2^2} = \sqrt{1+4+4} = \sqrt{9} = 3$
   $\|\mathbf{v}\|_1 = |1| + |-2| + |2| = 1 + 2 + 2 = 5$
   Note: $L_1 \geq L_2$ always (by Cauchy–Schwarz).
2. $\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|_2} = \frac{1}{3}(1, -2, 2) = \left(\frac{1}{3}, -\frac{2}{3}, \frac{2}{3}\right)$
   Verify: $\|\hat{\mathbf{v}}\|_2 = \sqrt{1/9 + 4/9 + 4/9} = \sqrt{9/9} = 1$ ✓
3. $\mathbf{v} = \mathbf{0}$ (the zero vector). The only vector with zero length.

Cosine Similarity: The Workhorse of NLP

Definition:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|} = \frac{\sum_i u_i v_i}{\sqrt{\sum_i u_i^2} \cdot \sqrt{\sum_i v_i^2}}$$

Properties:

- Range: $[-1, 1]$. $\cos = 1$: identical direction $\cos = 0$: orthogonal $\cos = -1$: opposite

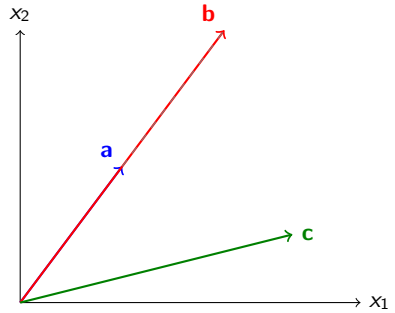
Numerical example:

$$\mathbf{u} = (1, 2, 3), \quad \mathbf{v} = (4, 5, 6) \qquad \cos(\mathbf{u}, \mathbf{v}) = \frac{32}{\sqrt{14} \cdot \sqrt{77}} = \frac{32}{32.83} \approx 0.974$$

In Finance Research

Li et al. (2021) use cosine similarity to expand seed dictionaries: find words whose embeddings are close to "teamwork," "innovation," etc.

Why Cosine over Euclidean Distance?



- **a** and **b** point in the **same direction** but different lengths
- Euclidean distance: $\|\mathbf{a} - \mathbf{b}\|$ is large (they're far apart!)
- Cosine similarity: $\cos(\mathbf{a}, \mathbf{b}) = 1.0$ (same direction = same meaning)

Cosine measures **direction**, ignoring **magnitude**. Two documents about banking may differ in length but share the same topic.

Practice: Cosine Similarity

Try these on your own (3 minutes):

1. Compute $\cos(\mathbf{u}, \mathbf{v})$ for $\mathbf{u} = (1, 0)$, $\mathbf{v} = (0, 1)$.
2. Compute $\cos(\mathbf{u}, \mathbf{v})$ for $\mathbf{u} = (3, 4)$, $\mathbf{v} = (6, 8)$. What do you notice?
3. Company A's embedding: $\mathbf{a} = (0.8, 0.6)$. Company B's embedding: $\mathbf{b} = (-0.8, -0.6)$. Compute $\cos(\mathbf{a}, \mathbf{b})$ and interpret.

Solution: Cosine Similarity

1. $\mathbf{u} \cdot \mathbf{v} = 0, \quad \|\mathbf{u}\| = 1, \quad \|\mathbf{v}\| = 1. \Rightarrow \cos(\mathbf{u}, \mathbf{v}) = \frac{0}{1 \times 1} = 0$ (orthogonal — completely unrelated dimensions).
2. $\mathbf{u} \cdot \mathbf{v} = 18 + 32 = 50. \quad \|\mathbf{u}\| = \sqrt{9 + 16} = 5, \quad \|\mathbf{v}\| = \sqrt{36 + 64} = 10.$
   $\cos = \frac{50}{5 \times 10} = 1.0. \quad \mathbf{v} = 2\mathbf{u}$: same direction, different magnitude. Cosine = 1 despite different lengths — this is why cosine ignores document length.
3. $\mathbf{a} \cdot \mathbf{b} = -0.64 - 0.36 = -1.0. \quad \|\mathbf{a}\| = 1, \|\mathbf{b}\| = 1.$
   $\cos(\mathbf{a}, \mathbf{b}) = -1.0$ (exactly opposite directions). The two companies have maximally dissimilar profiles.

Transpose and Symmetry

Transpose: Flip rows and columns. If $A \in \mathbb{R}^{m \times n}$, then $A^T \in \mathbb{R}^{n \times m}$.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad \Rightarrow \quad A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

Key rules:

- $(AB)^T = B^T A^T$ (reverse order!) $\qquad \mathbf{u}^T\mathbf{v} = \mathbf{v}^T\mathbf{u}$ (dot product is commutative) $\qquad (A^T)^T = A$

Symmetric matrix: $A = A^T$ (e.g., co-occurrence matrices in NLP)

$$X_{ij} = \text{count of word } j \text{ near word } i \quad \Rightarrow \quad X = X^T$$

In Word2Vec: The prediction score is a dot product $\mathbf{u}_o^T \mathbf{v}_c$, which is a 1×1 scalar.

Practice: Transpose

Try these on your own (2 minutes):

1. Compute A^T for $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$. What is the shape of A^T?
2. If $A \in \mathbb{R}^{5\times 3}$, what is the shape of $A^T A$? And AA^T?
3. Verify $(AB)^T = B^T A^T$ for $A = \begin{pmatrix} 1 & 2 \end{pmatrix}$, $B = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$.

Solution: Transpose

1. $A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} \in \mathbb{R}^{3\times 2}$ (rows $\leftrightarrow$ columns).
2. $A^T \in \mathbb{R}^{3\times 5}$, so $A^T A \in \mathbb{R}^{3\times 3}$ (like $X'X$ in OLS!).
   $AA^T \in \mathbb{R}^{5\times 5}$. Note: $A^T A \neq AA^T$ in general (different sizes).
3. $AB = \begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \end{pmatrix} = (11)$, so $(AB)^T = (11)$.

   $B^T A^T = \begin{pmatrix} 3 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = (11)$. ✓ They match!

Singular Value Decomposition (SVD)

Any matrix $M \in \mathbb{R}^{m \times n}$ can be factored as: $M = U\Sigma V^T$

- $U \in \mathbb{R}^{m \times m}$: left singular vectors $\Sigma \in \mathbb{R}^{m \times n}$: diagonal singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq 0$
- $V \in \mathbb{R}^{n \times n}$: right singular vectors (orthogonal columns)

Truncated SVD: Keep only top k singular values.

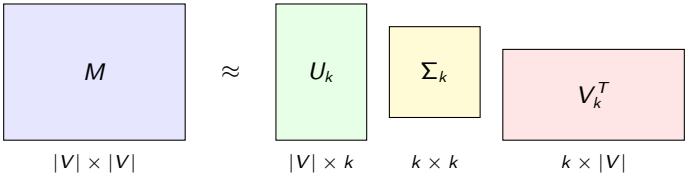
$$M \approx U_k \Sigma_k V_k^T$$

This gives the **best rank-k approximation** of M (minimizes $\|M - U_k \Sigma_k V_k^T\|$).

In NLP (Latent Semantic Analysis)

M = word–context co-occurrence matrix ($|V| \times |V|$). Truncated SVD $\rightarrow$ each word gets a k-dimensional vector. This is exactly what the CS224N Assignment 1 does!

SVD: Visual Intuition



Practical benefit: If $|V| = 500{,}000$ and $k = 300$:

- Original: 500,000-dimensional sparse vectors
- After SVD: 300-dimensional dense vectors
- Nearby vectors $\approx$ semantically similar words

Connection to Word2Vec: Levy & Goldberg (2014) showed that Word2Vec with negative sampling implicitly factorizes a shifted PMI matrix — similar to SVD on co-occurrence!

Practice: SVD

Try these on your own (2 minutes):

1. If $M \in \mathbb{R}^{100 \times 500}$ and we use truncated SVD with $k = 50$, what are the shapes of U_k, Σ_k, V_k^T?
2. Singular values are $\sigma = (10, 5, 1, 0.01)$. Which k captures most of the information? Why?
3. In NLP, if $|V| = 500{,}000$ and $k = 300$, what dimensionality is each word vector **before** and **after** SVD?

Solution: SVD

1. $U_k \in \mathbb{R}^{100\times 50}$, $\Sigma_k \in \mathbb{R}^{50\times 50}$ (diagonal), $V_k^T \in \mathbb{R}^{50\times 500}$.
   Verify: $(100 \times 50)(50 \times 50)(50 \times 500) = 100 \times 500$ ✓
2. $k = 3$ captures $\frac{10+5+1}{10+5+1+0.01} = \frac{16}{16.01} \approx 99.9\%$ of the total "energy" (sum of singular values). The 4th component adds almost nothing.
3. Before SVD: each word has a 500,000-dimensional sparse vector (one entry per context word).
   After SVD: each word has a 300-dimensional dense vector. This is the row of $U_k\Sigma_k$ (or equivalently U_k scaled by singular values).

The Softmax Function

Problem: We have raw scores (logits) $z_1, z_2, \ldots, z_{|V|}$ and need a probability distribution.
Softmax:

$$P(y = k) = \text{softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{|V|} e^{z_j}}$$

Properties:

- All outputs positive ($e^x > 0$) and sum to 1 (valid probability distribution)
- Larger $z_k \to$ larger probability (monotonic); "amplifies" differences

Example: $\mathbf{z} = (2.0, 1.0, 0.0)$

$$\text{softmax} = \left(\frac{e^2}{e^2 + e^1 + e^0}, \frac{e^1}{e^2 + e^1 + e^0}, \frac{e^0}{e^2 + e^1 + e^0} \right) \approx (0.67,\ 0.24,\ 0.09)$$

In Word2Vec

Softmax converts dot-product scores $\mathbf{u}_w^T \mathbf{v}_c$ into $P(\text{word} \mid \text{context})$.

Practice: Softmax

Try these on your own (2 minutes):

1. Compute softmax($[1, 1, 1]$). What do you notice about the result?
2. Compute softmax($[10, 0, 0]$) approximately. What happens when one logit is much larger?
3. If softmax gives $P = (0.70, 0.20, 0.10)$ for words ("loan", "rate", "bank"), which word is predicted as most likely? What is $\sum_i P_i$?

Solution: Softmax

1. All logits are equal, so $\text{softmax}([1,1,1]) = \left(\frac{e^1}{3e^1}, \frac{e^1}{3e^1}, \frac{e^1}{3e^1}\right) = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$.

   Equal logits $\Rightarrow$ **uniform distribution**. The e^1 cancels out.

2. $e^{10} \approx 22{,}026$, $\quad e^0 = 1$. Denominator $\approx 22{,}028$.

   softmax $\approx (0.9999,\ 0.00005,\ 0.00005)$. When one logit dominates, softmax becomes **nearly deterministic** — almost all probability mass on the largest logit.

3. "Loan" is most likely with $P = 0.70$.

   $\sum_i P_i = 0.70 + 0.20 + 0.10 = 1.0 \quad \checkmark$ (always sums to 1 by construction).

Derivatives: Rate of Change

Derivative of $f(x)$: the slope of f at point x.

$$f'(x) = \frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Common derivatives you will see:

$f(x)$	$f'(x)$	**Where it appears**
x^n	nx^{n-1}	Polynomial losses
e^x	e^x	Softmax, sigmoid
$\ln(x)$	$1/x$	Log-likelihood
$\sigma(x) = \frac{1}{1+e^{-x}}$	$\sigma(x)(1-\sigma(x))$	Negative sampling

Why Derivatives?

Training = minimizing a loss function. Derivatives tell us which direction to adjust parameters.

Practice: Derivatives

Try these on your own (2 minutes):

1. Compute $\frac{d}{dx}\left(3x^4 - 2x^2 + 5x - 1\right)$.
2. Compute $\frac{d}{dx}\left(e^{3x}\right)$. (*Hint:* use the chain rule with $u = 3x$.)
3. Compute $\frac{d}{dx}\left(\ln(x^2)\right)$.

Solution: Derivatives

1. Apply the power rule term by term:
   $\frac{d}{dx}(3x^4 - 2x^2 + 5x - 1) = 12x^3 - 4x + 5$
2. Let $u = 3x$, so $\frac{d}{dx}e^u = e^u \cdot \frac{du}{dx} = e^{3x} \cdot 3 = 3e^{3x}$.
3. Two equivalent approaches:
   Method 1: Simplify first. $\ln(x^2) = 2\ln(x)$, so $\frac{d}{dx}(2\ln x) = \frac{2}{x}$.
   Method 2: Chain rule. Let $u = x^2$: $\frac{d}{dx}\ln(u) = \frac{1}{u} \cdot \frac{du}{dx} = \frac{1}{x^2} \cdot 2x = \frac{2}{x}$. ✓

Partial Derivatives and Gradients

Partial derivative: Derivative with respect to one variable, holding others fixed.
Example: $f(x, y) = 3x^2y + 2xy^3 - 5x + 7$

$$\frac{\partial f}{\partial x} = 6xy + 2y^3 - 5 \qquad \frac{\partial f}{\partial y} = 3x^2 + 6xy^2$$

The Gradient: Collect all partial derivatives into a vector.

$$\nabla f = \begin{pmatrix} \partial f/\partial x \\ \partial f/\partial y \end{pmatrix}$$

Key property: The gradient ∇f points in the direction of **steepest increase** of f.

In Word2Vec

Parameters θ include all word vectors. $\nabla_\theta J(\theta)$ tells us how to adjust every word vector to reduce the loss.

Practice: Partial Derivatives and Gradients

Try these on your own (3 minutes):

1. $f(x, y) = x^2y + 3xy^2$. Compute $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$.
2. Evaluate the gradient ∇f at the point $(x, y) = (1, 2)$.
3. At $(1, 2)$, in which direction does f increase fastest?

Solution: Partial Derivatives and Gradients

1. Treat the other variable as a constant:
   $\frac{\partial f}{\partial x} = 2xy + 3y^2$ (differentiate w.r.t. x, treat y as constant)
   $\frac{\partial f}{\partial y} = x^2 + 6xy$ (differentiate w.r.t. y, treat x as constant)
2. At $(1, 2)$:
   $\frac{\partial f}{\partial x}\big|_{(1,2)} = 2(1)(2) + 3(4) = 4 + 12 = 16$
   $\frac{\partial f}{\partial y}\big|_{(1,2)} = 1 + 6(1)(2) = 1 + 12 = 13$
   $\nabla f\big|_{(1,2)} = \begin{pmatrix} 16 \\ 13 \end{pmatrix}$
3. In the direction of $\nabla f = (16, 13)$. The gradient always points toward steepest ascent. To *minimize* loss, we go the opposite direction: $-\nabla f$.

The Chain Rule: Foundation of Backpropagation

Chain rule: If $y = f(g(x))$, then:

$$\frac{dy}{dx} = \frac{dy}{dg} \cdot \frac{dg}{dx} = f'(g(x)) \cdot g'(x)$$

Example: $y = \ln(\sigma(x))$ where $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\frac{dy}{dx} = \frac{1}{\sigma(x)} \cdot \sigma(x)(1 - \sigma(x)) = 1 - \sigma(x)$$

Multivariate chain rule: If f depends on $\mathbf{z}$ which depends on $\mathbf{x}$:

$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial z_j} \cdot \frac{\partial z_j}{\partial x_i}$$

Why This Matters

Neural networks are compositions of functions. The chain rule lets us compute $\frac{\partial \text{Loss}}{\partial \text{any parameter}}$ — this is **backpropagation**.

Practice: Chain Rule

Try these on your own (3 minutes):

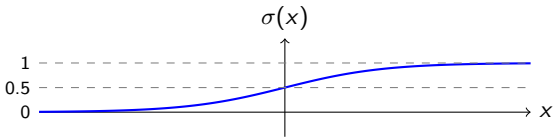
1. $y = (3x + 1)^5$. Compute $\frac{dy}{dx}$.
2. $y = e^{-x^2}$. Compute $\frac{dy}{dx}$.
3. In Word2Vec, the negative sampling loss for a positive pair is $J = -\ln(\sigma(z))$ where $z = \mathbf{u}_o^T \mathbf{v}_c$. Show that $\frac{dJ}{dz} = \sigma(z) - 1$.

Solution: Chain Rule

1. Let $u = 3x + 1$, so $y = u^5$.
   $\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} = 5u^4 \cdot 3 = 15(3x + 1)^4$
2. Let $u = -x^2$, so $y = e^u$.
   $\frac{dy}{dx} = e^u \cdot \frac{du}{dx} = e^{-x^2} \cdot (-2x) = -2x\, e^{-x^2}$
3. Chain of three functions: $J = -\ln(\sigma(z))$.
   $\frac{dJ}{dz} = \frac{dJ}{d\sigma} \cdot \frac{d\sigma}{dz} = \frac{-1}{\sigma(z)} \cdot \sigma(z)(1 - \sigma(z))$
   $= -(1 - \sigma(z)) = \sigma(z) - 1 \quad \checkmark$
   This elegant result makes Word2Vec gradient computation very efficient!

The Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Properties:

- Maps any real number to $(0, 1)$ — interpretable as a probability
- $\sigma(0) = 0.5, \quad \sigma(\infty) \to 1, \quad \sigma(-\infty) \to 0$
- Derivative: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ — max at $x = 0$, saturates at extremes

In Word2Vec negative sampling: $\sigma(\mathbf{u}_o^T \mathbf{v}_c)$ = probability that (c, o) is a real context pair.

Practice: Sigmoid Function

Try these on your own (2 minutes):

1. Compute $\sigma(0)$, $\sigma(100)$, and $\sigma(-100)$ (exact or approximate).
2. Show that $\sigma(-x) = 1 - \sigma(x)$. (*Hint:* multiply top and bottom of $\sigma(-x)$ by e^x.)
3. If $\sigma(\mathbf{u}_o^T \mathbf{v}_c) = 0.8$ for a word pair, interpret this value in Word2Vec negative sampling.

Solution: Sigmoid Function

1. $\sigma(0) = \frac{1}{1+e^0} = \frac{1}{2} = 0.5$

   $\sigma(100) = \frac{1}{1+e^{-100}} \approx \frac{1}{1+0} = 1$ (e^{-100} is astronomically small)

   $\sigma(-100) = \frac{1}{1+e^{100}} \approx 0$ (e^{100} is astronomically large)

2. $\sigma(-x) = \frac{1}{1+e^x} = \frac{e^{-x}}{e^{-x}(1+e^x)} = \frac{e^{-x}}{e^{-x}+1}$

   $1 - \sigma(x) = 1 - \frac{1}{1+e^{-x}} = \frac{e^{-x}}{1+e^{-x}}$ ✓ They are identical.

3. The model assigns 80% probability that (c, o) is a **true context pair** (not a random negative sample). The model is fairly confident these words co-occur naturally.

Matrix Calculus: The Shape Rule

Key rule: If loss J is a scalar and $\mathbf{W} \in \mathbb{R}^{m \times n}$, then:

$$\frac{\partial J}{\partial \mathbf{W}} \in \mathbb{R}^{m \times n} \quad \text{(same shape as } \mathbf{W}\text{!)}$$

Each entry tells you: "how much does J change if I nudge W_{ij}?"

Example: Embedding matrix $\mathbf{W} \in \mathbb{R}^{300 \times 50000}$

- $\nabla_{\mathbf{W}} J$ is also $300 \times 50{,}000$
- Entry (i, j): how the loss changes when you adjust the i-th dimension of the j-th word's embedding

For vectors: $\frac{\partial J}{\partial \mathbf{v}} \in \mathbb{R}^d$ when $\mathbf{v} \in \mathbb{R}^d$ — same shape.

PyTorch Does This For You

`loss.backward()` computes $\partial J / \partial \theta$ for every parameter θ automatically via the chain rule. You never compute gradients by hand in practice.

Practice: Matrix Calculus Shape Rule

Try these on your own (2 minutes):

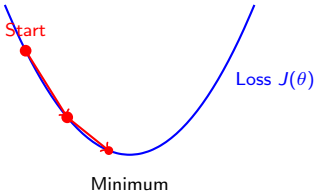
1. If J is a scalar loss and $W \in \mathbb{R}^{4\times 3}$, what is the shape of $\frac{\partial J}{\partial W}$?
2. If $\mathbf{v} \in \mathbb{R}^{300}$ is a single word vector, what is the shape of $\frac{\partial J}{\partial \mathbf{v}}$?
3. In BERT, one attention matrix is $W_Q \in \mathbb{R}^{768\times 64}$. How many individual gradient values does `loss.backward()` compute for this one matrix?

Solution: Matrix Calculus Shape Rule

1. $\frac{\partial J}{\partial W} \in \mathbb{R}^{4\times 3}$ — **same shape as** W.
   Each of the 12 entries tells you how much the loss changes when you nudge that specific parameter.
2. $\frac{\partial J}{\partial \mathbf{v}} \in \mathbb{R}^{300}$ — same shape as $\mathbf{v}$.
   This is a 300-dimensional vector. Entry i says: "how much does the loss change if I nudge the i-th dimension of this word's embedding?"
3. $768 \times 64 = 49{,}152$ gradient values for this single matrix.
   BERT-base has 12 attention heads $\times$ 12 layers $\times$ 4 matrices each $= 576$ such matrices, plus other parameters. Total: $\sim$110 million gradients computed per `backward()` call!

Gradient Descent: How Models Learn

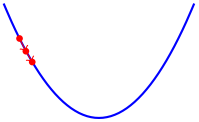
Goal: Find parameters θ that minimize loss $J(\theta)$.
Algorithm: (1) Initialize θ randomly. (2) Repeat: $\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$ where α is the **learning rate**.



Intuition: Walk downhill. The gradient tells you the steepest direction; the learning rate tells you how big a step to take.

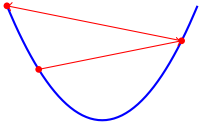
Learning Rate: Too Big vs. Too Small

Too small ($\alpha \ll$):



Converges **very slowly**. Might get stuck.

Too large ($\alpha \gg$):



Diverges! Overshoots the minimum.

Stochastic Gradient Descent (SGD):

- Don't compute gradient over entire dataset (too expensive)
- Instead, sample one example (or mini-batch) and update
- Noisier but **much faster** — standard for Word2Vec

Practice: Gradient Descent

Try these on your own (2 minutes):

1. Current parameter $\theta = 5$, learning rate $\alpha = 0.1$, gradient $\frac{\partial J}{\partial \theta} = 3$. What is θ after one gradient descent update?
2. If $\frac{\partial J}{\partial \theta} = 0$ at some θ^*, what does this mean geometrically?
3. Why might a larger learning rate converge faster initially but diverge later?

Solution: Gradient Descent

1. Apply the update rule: $\theta \leftarrow \theta - \alpha \frac{\partial J}{\partial \theta}$
   $\theta = 5 - 0.1 \times 3 = 5 - 0.3 = 4.7$
   The parameter moved in the *negative* gradient direction (downhill).
2. θ^* is a **critical point** where the loss surface is flat. In convex problems (like OLS), this is the unique minimum. In neural networks, it could be a local minimum or saddle point.
3. Large α takes big steps $\Rightarrow$ fast early progress when far from the minimum. But near the minimum, the loss surface curves sharply — large steps overshoot, bouncing back and forth across the valley. Solution: learning rate schedules (decay α over time).

Mini-Batch SGD and Epochs

Three flavors of gradient descent:

	Batch GD	SGD	Mini-batch SGD
Gradient on	All data	1 example	16–64 examples
Update quality	Accurate	Very noisy	Good balance
Speed per update	Slow	Fast	Fast

Mini-batch SGD is used in practice for everything: Word2Vec, BERT, GPT.

Epoch = one complete pass through the training data.

- Word2Vec: 5–20 epochs BERT fine-tuning: 3–5 epochs

Training loop:

1. Shuffle data
2. Split into mini-batches
3. For each batch: forward pass $\rightarrow$ loss $\rightarrow$ backward pass $\rightarrow$ update θ
4. Repeat for N epochs

Practice: Mini-Batch SGD and Epochs

Try these on your own (2 minutes):

1. A dataset has 10,000 training examples and batch size = 64. How many gradient updates occur per epoch?
2. With 5 epochs, how many **total** gradient updates?
3. Which is noisier: batch size 16 or batch size 256? Which computes faster per step?

Solution: Mini-Batch SGD and Epochs

1. $\lceil 10{,}000/64 \rceil = 157$ updates per epoch (156 full batches + 1 partial batch of 16 examples). Each update computes a gradient on 64 examples and adjusts θ.
2. $157 \times 5 = 785$ total gradient updates across all epochs.

   Note: each epoch sees the *same* data but in a different random order (shuffled).
3. **Batch size 16** is noisier — gradient is estimated from only 16 examples, so it has higher variance.

   Batch size 16 is faster per step (less computation per update).

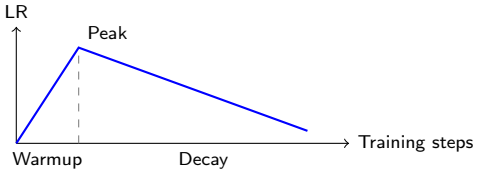
   Batch size 256 gives a more accurate gradient estimate but takes longer per step. In practice, batch size 32–64 is a common sweet spot.

Adam Optimizer and Learning Rate Schedules

Adam (Kingma & Ba, 2015) — the default optimizer for transformers:

- **Momentum:** Running average of past gradients (smooths noise)
- **Adaptive rates:** Each parameter gets its own learning rate
- In practice: `optimizer = torch.optim.Adam(params, lr=2e-5)`

Learning rate schedules — warmup then decay:



Why Warmup?

Early gradients are noisy (random initialization). Small steps first, then larger steps once the model is in a reasonable region. Critical for BERT fine-tuning.

Practice: Adam and Learning Rate Schedules

Try these on your own (2 minutes):

1. Why does Adam use "adaptive" learning rates (one per parameter) instead of a single global α?
2. In warmup, we start with lr ≈ 0 and increase to peak. Why not start at the peak learning rate?
3. BERT fine-tuning uses lr $= 2 \times 10^{-5}$, 3 epochs, 10% warmup. If there are 1,000 total training steps, how many are warmup steps?

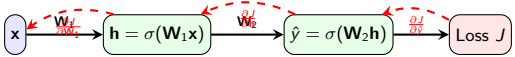
Solution: Adam and Learning Rate Schedules

1. Different parameters may have very different gradient magnitudes. For example, rare-word embeddings get updated infrequently (sparse gradients) while bias terms get updated every step. A single α that works for one may be too large or too small for another. Adam adapts automatically.
2. Early in training, parameters are randomly initialized, so gradients are **unreliable** — they point in roughly random directions. A large learning rate would cause large, misguided updates. Starting small lets the model "warm up" to a reasonable region before taking bigger steps.
3. 10% of 1,000 = **100** warmup steps.

   During these 100 steps, the learning rate linearly increases from $\sim$0 to 2×10^{-5}. Then it decays over the remaining 900 steps.

Backpropagation: Chain Rule on a Computational Graph

Forward pass: compute output and loss. **Backward pass:** compute gradients via chain rule, from loss back to parameters.



Chain rule in action:

$$\frac{\partial J}{\partial \mathbf{W}_1} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{W}_1}$$

Key Insight

Backprop = apply the chain rule from the loss backward through each layer. PyTorch's `autograd` does this automatically.

Practice: Backpropagation

Try these on your own (2 minutes):

1. In the computational graph $x \to h \to y \to J$, write $\frac{\partial J}{\partial x}$ using the chain rule.
2. Given: $\frac{\partial J}{\partial y} = 2$, $\frac{\partial y}{\partial h} = 0.5$, $\frac{\partial h}{\partial x} = 3$. Compute $\frac{\partial J}{\partial x}$.
3. Why does backpropagation go from the loss *backward* (not forward from inputs)?

Solution: Backpropagation

1. By the chain rule:
   $\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial x}$
   Each term is a "local gradient" at one node in the graph.
2. $\frac{\partial J}{\partial x} = 2 \times 0.5 \times 3 = 3$
   Interpretation: if we increase x by a tiny amount ϵ, the loss J increases by approximately 3ϵ.
3. Each layer's gradient **depends on the gradient from the layer above**. We need $\frac{\partial J}{\partial y}$ before we can compute $\frac{\partial J}{\partial h}$ (since $\frac{\partial J}{\partial h} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial h}$). So we must start at the loss and work backward. This is why it's called *back*-propagation.

Vanishing and Exploding Gradients

Problem: In deep networks, gradients are *products* of many terms (chain rule).

Vanishing gradients:

- Each factor < 1 (e.g., sigmoid derivative ≤ 0.25)
- $0.25^{10} \approx 10^{-6}$ — early layers barely learn

Exploding gradients:

- Each factor > 1
- Gradient grows exponentially — training diverges

Solutions (all appear in transformers / BERT):

- **ReLU activation:** $\text{ReLU}(x) = \max(0, x)$. Derivative is 0 or 1 — no shrinking!
- **Residual connections:** $\mathbf{h} = f(\mathbf{x}) + \mathbf{x}$. Gradient flows directly through the "+"
- **Layer normalization:** Keeps activations in a stable range
- **Gradient clipping:** Cap gradient norm if it exceeds a threshold

Practice: Vanishing and Exploding Gradients

Try these on your own (2 minutes):

1. If each layer multiplies the gradient by 0.2, what is the gradient magnitude at layer 1 in a 5-layer network? (Assume gradient starts at 1.0 at the output.)
2. $\text{ReLU}(x) = \max(0, x)$. What is its derivative for $x > 0$? For $x < 0$?
3. In a residual connection $\mathbf{h} = f(\mathbf{x}) + \mathbf{x}$, compute $\frac{\partial \mathbf{h}}{\partial \mathbf{x}}$. Why does this help?

Solution: Vanishing and Exploding Gradients

1. $0.2^5 = 0.2 \times 0.2 \times 0.2 \times 0.2 \times 0.2 = 0.00032$

   The gradient has **vanished** to 0.03% of its original value! Layer 1 receives almost no learning signal. This is why deep sigmoid networks are hard to train.

2. For $x > 0$: $\text{ReLU}'(x) = 1$ (gradient passes through unchanged).

   For $x < 0$: $\text{ReLU}'(x) = 0$ (gradient is blocked, the "dead ReLU" problem).

   Key: for active neurons ($x > 0$), the derivative is exactly 1 — no shrinking!

3. $\frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} + \boldsymbol{I}$

   The "$+\boldsymbol{I}$" (identity) ensures the gradient is **at least 1** in each direction, even if $\frac{\partial f}{\partial \mathbf{x}}$ is small. This is the key insight of ResNets and Transformers.

Loss Functions: What We Optimize

Cross-entropy loss (negative log-likelihood):

$$J = -\log P(\text{correct answer})$$

Why log?

- Turns products into sums (numerically stable)
- Penalizes confident wrong predictions harshly
- $P = 1 \Rightarrow J = 0$ (perfect), $\quad P \to 0 \Rightarrow J \to \infty$ (terrible)

Word2Vec Skip-gram Loss:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j}|w_t;\theta)$$

Translation: For each word in the corpus, maximize the probability of its neighboring words.

Optimization Landscape

Good parameters $\theta \to$ small loss $\to$ high probability assigned to actual context words.

Practice: Loss Functions

Try these on your own (2 minutes):

1. The model assigns $P(\text{correct}) = 0.9$. Compute the cross-entropy loss $J = -\ln(P)$.
2. Now $P(\text{correct}) = 0.01$. Compute J. How does it compare to Q1?
3. In Word2Vec, if $P(w_{t+1} \mid w_t) = 0.001$ for the actual next word, is the model doing well? What should happen to this value during training?

Solution: Loss Functions

1. $J = -\ln(0.9) \approx 0.105$

   A small loss — the model is quite confident in the correct answer.

2. $J = -\ln(0.01) \approx 4.605$

   44× worse than Q1! Cross-entropy penalizes low-confidence predictions *harshly*. Going from $P = 0.9$ to $P = 0.01$ increases the loss by a factor of 44.

   This is by design: the log function makes the penalty grow very fast as $P \to 0$.

3. The model is doing **poorly** — assigning only 0.1% probability to the actual context word out of a vocabulary of (say) 50,000 words.

   During training, gradient descent should **increase** $P(w_{t+1} \mid w_t)$ by adjusting the word vectors so that the true context word's dot product with the center word grows larger.

From Econometrics to Machine Learning

You already know the core idea. ML reframes it with different tools.

	Econometrics	**Machine Learning**
Goal	Estimate β, test hypotheses	Minimize prediction loss
Model	$y = X\beta + \varepsilon$	$\hat{y} = f(X; \theta)$
"Training"	OLS: $\hat{\beta} = (X'X)^{-1}X'y$	Gradient descent (no closed form)
Evaluation	R^2, t-stats, F-test	Out-of-sample loss, accuracy, F1
Overfitting	Adjusted R^2, BIC	Train/val/test split, early stopping
Complexity	Add/remove variables	Regularization, dropout

Why No Closed Form?

Neural networks (Word2Vec, BERT, GPT) are highly nonlinear — the loss surface has no analytical solution. Must use iterative optimization (gradient descent).

Supervised Learning: The Universal Template

Almost everything in this course follows one pattern:

1. **Data:** Pairs (X_i, y_i) — input and desired output
2. **Model:** $\hat{y}_i = f(X_i; \theta)$ — parameterized function
3. **Loss:** $J(\theta) = \frac{1}{N} \sum_i L(\hat{y}_i, y_i)$ — scalar measuring error
4. **Optimize:** $\theta \leftarrow \theta - \alpha \nabla_\theta J$ — gradient descent

Course examples:

Method	**Input** X	**Label** y	**Loss**
Word2Vec	Center word	Context word	Cross-entropy
BERT pretrain	Sentence with [MASK]	Masked word	Cross-entropy
BERT fine-tune	Document text	Sentiment label	Cross-entropy

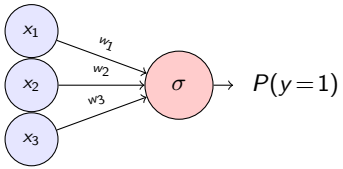
Observation

Cross-entropy loss appears everywhere. The inputs and labels change; the framework stays the same.

Logistic Regression = A Single Neuron

You know logistic regression from econometrics. It is literally a one-neuron neural network.

$$P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}}$$



A neural network = many neurons stacked in layers.
BERT fine-tuning:

- BERT encodes a document into a 768-dim vector
- A logistic regression head maps that vector to $P(\text{positive sentiment})$
- Same math, much richer features

Train / Validation / Test Split

Training set

- Learn parameters θ
- Model sees labels
- Can reuse many times

Validation set

- Tune hyperparameters
- Detect overfitting
- Never train on this

Test set

- Final evaluation
- Touch only ONCE
- Report this number

Overfitting: Model memorizes training data but fails on new data.

- Training loss $\downarrow$ but validation loss $\uparrow$ — stop training (early stopping)
- **Econometrics analogy:** In-sample R^2 is always optimistic; out-of-sample R^2 matters

Evaluation Metrics for Classification

For BERT fine-tuning and misstatement prediction (e.g. TAR 2025), you need:

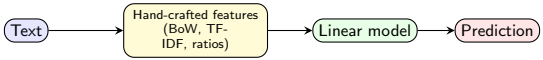
Metric	Definition
Accuracy	$\frac{\text{Correct predictions}}{\text{Total predictions}}$
Precision	$\frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$ (of those we flagged, how many are real?)
Recall	$\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$ (of real positives, how many did we catch?)
F1 Score	$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$ (harmonic mean)

Why not just accuracy?

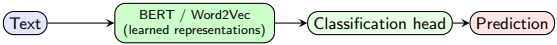
- If 99% of firms are clean, predicting "no misstatement" always gives 99% accuracy!
- **F1 balances precision and recall** — standard for imbalanced classification

The Big Idea: Representation Learning

Traditional ML: Human designs features → model learns weights



Deep learning: Model learns *both* features (representations) AND weights



Course Arc

Lecture 1 (Word2Vec): learn word representations automatically from co-occurrence.
Lecture 9 (BERT): learn *contextual* representations that encode meaning in context.

Course Tech Stack

Tool	Purpose	When
Python 3.10+	Programming language	Always
NumPy	Arrays, linear algebra	Always
pandas	DataFrames, data wrangling	Data lectures
matplotlib	Plotting	Visualization
gensim	Word2Vec, GloVe	NLP lectures
scikit-learn	SVD, TF-IDF, evaluation	NLP + ML
PyTorch	Deep learning	Transformers, BERT
Jupyter	Interactive notebooks	Assignments
Git/GitHub	Version control	Projects

Environment setup:

```
conda create -n bphd8063 python=3.12
conda activate bphd8063
pip install numpy pandas gensim torch scikit-learn matplotlib jupyter
```

Python Data Types and Control Flow

```python
# Basic types
x = 42              # int
r = 0.05            # float
name = "JPMorgan"   # str
is_public = True    # bool

# Lists (ordered, mutable)
tickers = ["AAPL", "JPM", "MSFT", "GS"]
returns = [0.02, -0.01, 0.03, 0.005]

# Dictionaries (key-value pairs)
firm = {"ticker": "AAPL", "sector": "Tech", "mcap": 2800}

# Control flow
for t in tickers:
    print(t)

# Conditional
if r > 0:
    print("Positive return")
elif r == 0:
    print("Zero return")
else:
    print("Negative return")
```

List Comprehensions: Pythonic Patterns

```python
# Instead of:
tech_tickers = []
for c in companies:
    if c["sector"] == "Technology":
        tech_tickers.append(c["ticker"])

# Write:
tech_tickers = [c["ticker"] for c in companies
                if c["sector"] == "Technology"]

# Flatten a list of lists (useful for NLP corpora)
corpus = [["the", "bank"], ["raised", "rates"]]
all_words = [w for doc in corpus for w in doc]
# -> ["the", "bank", "raised", "rates"]

# Count words per sector
from collections import Counter
sector_counts = Counter(c["sector"] for c in companies)
# -> Counter({"Technology": 3, "Finance": 2})
```

Performance

List comprehensions are faster than `for` loops in Python — prefer them.

NumPy: Arrays and Vectorized Operations

```python
import numpy as np

# Create arrays
u = np.array([1, 2, 3])            # 1-D vector
v = np.array([4, 5, 6])
W = np.random.randn(300, 50000)    # Embedding matrix

# Element-wise operations (vectorized -- no loops!)
w = u + v           # [5, 7, 9]
w = u * v           # [4, 10, 18]  (NOT matrix multiply)
w = u ** 2          # [1, 4, 9]

# Shape matters!
print(u.shape)      # (3,)   -- 1-D array
A = np.array([[1, 2], [3, 4]])
print(A.shape)      # (2, 2)

# Reshape
u_col = u.reshape(-1, 1)    # (3, 1) column vector
u_row = u.reshape(1, -1)    # (1, 3) row vector
```

NumPy: Linear Algebra Operations

```python
import numpy as np

u = np.array([1.0, 2.0, 3.0])
v = np.array([4.0, 5.0, 6.0])

# Dot product
dot = np.dot(u, v)               # 32.0
dot = u @ v                      # Same thing

# Norms
norm_u = np.linalg.norm(u)      # sqrt(14) = 3.742

# Cosine similarity
def cosine_sim(u, v):
    return np.dot(u, v) / (np.linalg.norm(u)
                           * np.linalg.norm(v))
print(cosine_sim(u, v))          # 0.9746

# Matrix multiply
A = np.array([[1, 0, 2], [3, 1, -1]])    # (2, 3)
B = np.array([[2, 1], [0, -1], [4, 3]]) # (3, 2)
C = A @ B                                 # (2, 2)
print(C)    # [[10, 7], [2, -1]]
```

NumPy: Broadcasting

Broadcasting: NumPy automatically expands dimensions for element-wise operations.

```
import numpy as np

# Normalize each row of a matrix to unit length
M = np.array([[3.0, 4.0],
              [1.0, 0.0],
              [0.0, 5.0]])

# Row norms: shape (3,)
norms = np.linalg.norm(M, axis=1)          # [5, 1, 5]

# Reshape to (3, 1) so division broadcasts across columns
M_normalized = M / norms[:, np.newaxis]
# [[0.6, 0.8], [1.0, 0.0], [0.0, 1.0]]
```

Rule of thumb:

- `(3,)` + `(3,)` works `(3,1)` + `(1,4)` → `(3,4)` `(3,)` + `(4,)` fails

You will use this in the word vectors assignment to normalize embedding matrices.

Putting It Together: Cosine Similarity Matrix

Real task: Given word embeddings, compute pairwise similarity for all words.

```
import numpy as np

# Suppose we have embeddings for 5 words, 300-dim each
W = np.random.randn(5, 300)

# Step 1: Normalize each word vector to unit length
norms = np.linalg.norm(W, axis=1, keepdims=True)
W_norm = W / norms

# Step 2: Cosine similarity matrix = dot products
#   (5, 300) @ (300, 5) -> (5, 5)
sim_matrix = W_norm @ W_norm.T

print(sim_matrix.shape)   # (5, 5)
print(sim_matrix[0, 1])   # cosine sim between word 0 and 1
# Diagonal is all 1.0 (each word is identical to itself)
```

This is what gensim does

`model.wv.most_similar("bank")` computes cosine similarity between "bank" and every word, then returns the top matches.

Quick Pandas Preview

```python
import pandas as pd

# Load data
df = pd.read_csv("stock_returns.csv")
# Columns: ticker, date, ret, sector

# Filter
big_moves = df[df["ret"].abs() > 0.05]

# Group-by aggregation
sector_stats = (df.groupby("sector")["ret"]
                  .agg(["mean", "std"]))

# Merge datasets (like SQL JOIN)
merged = pd.merge(returns_df, fundamentals_df,
                  on=["ticker", "year"], how="inner")
```

For this course:

- Lectures 3–6 (data infrastructure, panels) will use pandas/polars heavily
- Today: just know that DataFrames exist and are like spreadsheets in Python

Cheat Sheet: Math $\rightarrow$ NLP Connection

Concept	Where It Appears in This Course
Vectors in $\mathbb{R}^d$	Word embeddings, BERT hidden states
Dot product $\mathbf{u}^T\mathbf{v}$	Word2Vec similarity score, attention
Cosine similarity	Dictionary expansion (Li et al., Wu), evaluation
Matrix multiplication	Embedding lookup, attention (QK^T), neural layers
SVD	Co-occurrence $\rightarrow$ dense embeddings (LSA)
Softmax	$P(\text{word}\|\text{context})$ in Word2Vec, attention weights
Sigmoid $\sigma(x)$	Negative sampling, logistic regression head
Chain rule / backprop	Training any model (Word2Vec, BERT, GPT)
Adam optimizer	Default for transformer fine-tuning
Train/val/test split	Every experiment in Lectures 5–13
Precision, recall, F1	BERT classification, misstatement prediction

Questions?

Thank you!

Dr. (James) Yunying Huang
Fordham Gabelli School of Business